University of Ljubljana
Faculty of Computer and Information Science

Martin Frlin Novak

Developing Software Tools for In Situ Cognitive Load Estimation

bachelor's thesis

first cycle university study programme
computer and information science

Supervisor: doc. dr. Veljko Pejović

Ljubljana, 2019

Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Martin Frlin Novak

Razvoj programskih orodij za oceno kognitivne obremenitve

diplomsko delo

univerzitetni študijski program
prve stopnje
računalništvo in informatika

Mentor: doc. dr. Veljko Pejović

Ljubljana, 2019

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Faculty of Computer and Information Science issues the following thesis:

Theme of the thesis:

The thesis should conceptually design a protocol and relevant methods for exposing a user to tasks demanding various cognitive efforts. The user's performance on these tasks should be captured together with physiological signals that may relate to cognitive load (heart rate variability, skin conductance, and others). Finally, for gauging the user's cognitive load, the thesis should implement a secondary task that the user should perform in parallel to the primary task. Where possible, the thesis should rely on already verified methodologies and, where needed, implement original solutions. The protocol and the methods should be demonstrated in a small-scale lab experiment.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Naloga naj konceptualno razvije protokol in relevantne metode, ki izzovejo uporabnika z nalogami, katere zahtevajo različne stopnje kognitivnega napora. Uporabnikovo uspešnost pri teh nalogah naj se zajame skupaj z njegovimi fiziološkimi signali, za katere se verjame, da so korelirani s kognitivno obremenitvijo (srčni utrip, prevodnost kože in podobno). Naloga naj razvije sekundarno nalogo, ki jo uporabnik rešuje vzporedno s primarnimi nalogami. Namen te sekundarne naloge naj bo ocenjevanje uporabnikove kognitivne obremenitve. Kjer je mogoče naj se naloga opira na že razvite in preverjene metodologije in kjer je potrebno, razvije lastne rešitve. Protokol in metode naj se demonstrirajo na manjši populaciji v laboratorijskem testu z uporabniki.

Contents

Abstract

Povzetek

Razširjen povzetek

1 Introduction **1**

2 Related Work **3**

3 Cognitive Load Experimentation Framework **7**

3.1 Framework Components 8

3.1.1 N-back Test 8

3.1.2 Hexaco Personality Test 8

3.1.3 Cognitive Load Test 9

3.1.4 Secondary Task 10

3.1.5 Android Application for Physiological Data Collection 11

3.1.6 Demographics Questionnaire 11

4 Design and Implementation of Framework Components **13**

4.1 N-back Test 14

4.1.1 Description 14

4.1.2 Implementation 14

4.1.3 Test Results 16

4.2 Hexaco Test 17

4.2.1 Description 17
4.2.2 Implementation 17
4.2.3 Test Results 21
4.3 Secondary Task 21
4.3.1 Description 21
4.3.2 Design and Ideas 21
4.3.3 Implementation 22
4.3.4 Test Result 23

5 User Study **25**
5.1 Background and Overview 25
5.2 Test Protocol 27
5.3 Experiences 27

6 Conclusion and Future Work **31**
6.1 Lessons Learned 32
6.2 Future Work 32
6.3 Acknowledgements 33

Bibliography **34**

List of Abbreviations

abbrevation	full
HR	heart rate
BVP	blood volume pulse
GSR	galvanic skin response
ST	skin temperature
RR interval	time between successive R points in QRS complex
EEG	electroencephalography
ECG	electrocardiography
MVC	model view controller
ORM	object-relational manager
SQL	structured query language

Abstract

Title: Developing Software Tools for In Situ Cognitive Load Estimation

Author: Martin Frlin Novak

The thesis is motivated by the problem of fragmented attention. Multitasking is useful, however, uncontrolled interruptions coming from a large number of mobile devices and applications lead to poor task performance, frustration and other negative consequences. There have been attempts to control technology-induced interruptions. For example, by delaying notifications until more appropriate times. These moments are often characterized by low cognitive load of a user. However, inferring cognitive load is challenging. To date, the only options we have rely on rather intrusive techniques, such as pupil dilation measurements, requiring expensive sensing equipment, etc. The option we explore is whether cognitive load can be inferred through cheap, off-the-shelf devices, such as wearable computers (fitness wristbands, smart watches, and similar). In this thesis we present the design and implementation of an experimentation environment for cognitive load estimation. Our setup consists of a desktop application that presents the user with tasks of varying difficulty, another desktop application that runs in parallel which, through a side-task, aims to gauge the user's cognitive engagement, a wearable sensor that through a mobile phone app collects data about the user's physiological signals, and pre- and post-test questionnaires that evaluate the user's overall cognitive capacity and personality. In the thesis, we provide detailed explanations of the testing environment, especially the parts that

were originally developed for the thesis – the secondary task desktop application, the web-based pre/post questionnaires and tests, and the database orchestration – and report on the experiences with real-world experiments conducted with 27 volunteers.

Keywords: cognitive load, test environment, study design.

Povzetek

Naslov: Razvoj programskih orodij za oceno kognitivne obremenitve

Avtor: Martin Frlin Novak

Tema diplomske naloge izvira iz problema razdrobljene pozornosti. Večopravilnost je koristna, vendar nekontrolirane motnje, ki prihajajo iz raznih mobilnih naprav, vodijo do slabih rezultatov pri delu in frustracij. Zato je zaželjeno, da se te motnje kontrolirajo in obvestila prestavijo na kasnejši, bolj primeren čas. Na čas, ko človek ni v stanju visoke kognitivne obremenitve. Kar je težko izvesti, saj je detekcija stopnje kognitivne obremenitve izziv. Trenutni načini detekcije temeljijo na invazivnih tehnikah, kot na primer merjenje razširjenosti zenic, in potrebujejo specializirano in drago opremo. Način, ki ga preverjamo mi, pa temelji na poceni in vedno bolj razširjenih napravah, kot so fitnes zapestnice, pametne ure in podobno. V diplomski nalogi predstavljamo načrtovanje in implementacijo testirnega okolja za merjenje kognitivne obremenitve. Okolje sestavljajo računalniška aplikacija, ki uporabniku prikazuje naloge različnih težavnosti, vzporedna aplikacija, ki proži vzporedno nalogo, ki služi za ocenjevanje obremenitve, fitnes zapestnica, ki skozi mobilni telefon zbira podatke o uporabnikovih fizioloških parametrih in vprašalnikov, ki ocenijo uporabnikove kognitivne sposobnosti in osebnost. V diplomi je podrobno predstavljeno celotno okolje s posebnim poudarkom na delih, ki so bili razviti posebej za raziskavo - aplikacija za vzporedno nalogo, spletna aplikacija za ocenjevanje kognitivnih sposobnosti in osebnosti ter orkestracija podatkov - kot tudi izkušnje z izvajanjem testov na 27 uporabnikih.

Ključne besede: kognitivna obremenitev, okolje za izvajanje testov, načrtovanje študije.

Razširjen povzetek

Pozornost in koncentracija sta zelo dragoceni sposobnosti, ki pa zaradi nenehnih motenj in prekinjanj, vedno hitreje pešata. V informacijski dobi smo z napravami vedno povezani v splet in s tem tudi bolj dovzetni za motnje. Informacije so potisnjene k nam in povprečno dobimo 45 do 100 obvestil na dan. Naprave, ki nam dostavljajo obvestila, se ne ozirajo na to kaj delamo in kako zelo skoncentrirani smo na delo, zato dostikrat pride do izgube koncentracije, ko nas obvestila zmotijo.

Naprave bi lahko zaznale našo kognitivno obremenitev in se na podlagi tega odločile, da je obvestilo bolje zakasniti. To bi izboljšalo produktivnost uporabnikov, prišlo bi do manj napak pri delu in do boljšega počutja, saj je konstantna menjava miselenega konteksta zelo naporna.

Pri pregledu dosedanjega dela smo ugotovili, da se za zaznavanje kognitivne obremenjenosti uporablja specializirana in draga oprema. Raziskovalci uporabljajo magnetno resonanco, sledenje očem in merjenje razširjenosti zenice, EEG in podobno. Ti načini niso prenosljivi na veliko število uporabnikov, zato se naše delo osredotoča na poceni in lahko dostopne naprave, kot so fitnes zapestnice, pametne ure in mobilni telefoni. Naš cilj je preveriti, če je tudi s temi napravami možno zaznati kognitivno obremenitev uporabnika.

S tem ciljem smo razvili testirno okolje za zaznavo kognitivne obremenitve in zasnovali študijo ter izvedli teste na 27ih uporabnikih. Okolje je sestavljeno iz več aplikacij, ki so bile razvite oziroma prilagojene našim potrebam. Aplikacije z nalogami izzovejo uporabnika, da preide v stanje visoke kognitivne obremenitve, obenem pa se preko fitnes zapestnice pobirajo uporabnikovi

fiziološki znaki. Vsi uporabnikovi podatki se povežejo med sabo z identifikacijskim nizom in se shranijo, kjer so potem lahko dostopni za različne analize.

Okolje je sestavljeno iz naslednjih aplikacij:

Cognitive Load Test je aplikacija, ki smo jo odkrili pri pregledu podobnih študij in smo jo prevedli v slovenski jezik ter adaptirali, da se poveže z našimi drugimi aplikacijami. Aplikacija sestoji iz šest tipov nalog, ki kognitivno obremenijo uporabnika na različne načine. Vsak tip nalog ima tri stopnje težavnosti in po vsaki nalogi je NASA-TLX vprašalnik, kjer uporabnik subjektivno ovrednoti težavnost naloge in svoje občutke ob reševanju naloge. Aplikacija zapisuje svoje stanje v datoteko, iz katere potem dobimo uporabnikove odgovore, čas reševanja in ostale podatke, ki bi nas utegnili zanimati.

Secondary Task je aplikacija, ki smo jo razvili z namenom, da vzporedno s subjektivnim NASA-TLX vprašalnikom ovrednoti težavnost nalog iz *Cognitive Load Test* aplikacije. To stori tako, da med reševanjem nalog naključno sproži vzporedno nalogo. Naloga izgleda tako, da se na ekranu pokaže osenčen kvadrat, ki je na začetku skoraj prozoren in postopoma temni, dokler ni popolnoma črn. Uporabnik mora čim hitreje zaznati kvadrat in z računalniško miško klikniti nanj. Meri se reakcijski čas s predpostavko, da se ob reševanju težjih nalog, reakcijski čas podaljša.

N-back Test je spletna aplikacija, ki smo jo razvili z namenom, da preverimo kognitivno kapaciteto uporabnikov. Aplikacija deluje kot igra, kjer se v mreži velikosti 3 krat 3 na določen interval naključno obarva eno kvadratno polje. Uporabnik si mora zapomniti, katera polja so se obarvala, in zaznati, kadar se obarva isto polje kot N intervalov nazaj. Po koncu testa nam aplikacija pove, koliko krat je uporabnik pravilno zaznal pojav, koliko krat ga ni zaznal in koliko krat ga je napačno zaznal. Test je težji s povečevanjem števila N, saj je potrebno v spominu ohranjati lokacije obarvanih polj za N intervalov nazaj in z vsakim novim intervalom osvežiti lokacije v spominu.

Hexaco Personality Test je spletna aplikacija, ki smo jo razvili z na-

menom, da dobimo podatke o uporabnikovi osebnosti. Hexaco je osebnostni vprašalnik, kjer uporabnik odgovarja, če se strinja ali ne strinja s trditvami o njegovi osebnosti. Uporabnik mora za vsako od 60 trditev odgovoriti z eno izmed 5ih stopenj strinjanja, ki so razporejene od močno se strinjam do močno se ne strinjam. Odgovori se nato ovrednotijo in dobimo oceno kako močno so zastopani določeni vidiki osebnosti (naprimer poštenost, vedoželjnost, ustvarjalnost, ...) pri uporabniku.

Android aplikacija za pridobivanje fizioloških podatkov je aplikacija, ki so jo razvili na Inštitutu Jožef Štefan. Aplikacija je nameščena na mobilni telefon in se poveže s fitnes zapestnico Microsoft Band 2. V aplikacijo vneseš identifikacijski niz uporabnika in aplikacija začne pobirati podatke iz zapestnice ter jih pošilja v bazo na strežniku. Podatki iz zapestnice so fiziološke značilnosti uporabnika kot so srčni utrip, prevodnost kože in temperatura kože. Tile podatki so potem na voljo za analizo in ustvarjanje statističnih modelov, ki na podlagi podatkov ocenjujejo, kako močno je uporabnik kognitivno obremenjen.

Vsi podatki iz aplikacij so povezani preko identifikacijskega niza uporabnika in časa, zato morata biti uri na mobilnem telefonu in računalniku, kjer se izvajajo testi, usklajeni. Podatkom iz aplikacije se dodajo še demografski podatki uporabnika, ki jih uporabnik izpolni v vprašalniku na papirju. Za kompletno testirno okolje so seveda potrebne še prej omenjene naprave. Računalnik, kjer uporabnik rešuje teste, mobitel povezan z Microsoft Band 2 zapestnico in strežnik, na katerem tečejo spletne aplikacije in se shranjujejo podatki.

Okolje smo stestirali z izvajanjem testov v študiji, za katero smo zasnovali pridobivanje podatkov. Testi so se izvajali v prostorih Fakultete za Računalništvo in Informatiko ob spremstvu nadzornika, ki je sledil predpisanemu protokolu, tako da so bili testi karseda konsistentni. Uporabniki so najprej izpolnili demografski vprašalnik, rešili dve težavnosti *N-back* testa (N=2 in N=3) ter rešili *Hexaco* osebnostni test. Preden so začeli, so bili opremljeni z zapestnico, ki je pobirala fiziološke podatke med reševanjem.

Med posameznimi testi so imeli 3 minutne premore, da se odpočijejo in da se fiziološki znaki spet normalizirajo. Potem je sledil drugi del, v katerem so reševali naloge v *Cognitive Load Test* aplikaciji, vzporedno pa se je naključno prožil *Secondary Task*. Tudi tukaj so med posameznimi nalogami 3 minute počivali.

Preko e-email novičnikov fakultete in socialnih omrežij smo pridobili 27 prostovoljcev, ki so šli skozi teste. Ob izvajanju testov smo naleteli na par hroščev v aplikacijah in tehnične težave, zato smo dobili uporabne podatke samo od 21 uporabnikov. Težave smo odpravili in testirno okolje se je izkazalo za uporabno, saj sta z njegovo uporabo nastala dva objavljena članka.

Chapter 1

Introduction

Attention is the most precious, yet the most overused resource in the information age. Information is pushed to us via mobile devices, which enable us that we are always connected and make us available wherever we are. It is estimated that in year 2019 there will be more than 5 billion mobile phone users which is 67% of the world's population [2]. On average, each phone user receives 45 [1] to 100 [19] push notifications per day. This means we are vulnerable to interruptions, forcing us to context switch and take away concentration from the task at hand.

Our attention is fragmented because devices do not interact with us in a considerate way. This leads to problems with task completion, errors while executing a task and user frustration [8]. We can see from this that managing interruptions is beneficial for users and would increase efficiency and lower error rates.

There have been attempts to control computing technology-induced interruptions. For example, by delaying notifications until more appropriate times [16]. These moments are often characterized by low cognitive load of the user [6]. However, inferring cognitive load is challenging. To date, the only options for data collection we have rely on rather intrusive techniques, such as pupil dilation measurements and require expensive sensing equipment [17]. Furthermore, one needs to process the data and construct appropriate ma-

chine learning models and evaluate them. But as mobile devices are getting smarter, maybe they can help us with this problem. If we can detect state of high cognitive load with accessible consumer grade devices such as mobile phones and fitness wristbands, then we can delay notifications and prevent interruptions, therefore prolonging users' concentration and make them more productive and improve their quality of life. Here we tackle the first problem of data acquisition with a framework for cognitive load experimentation.

In this thesis, we adapt experimentation environment for cognitive load estimation based on commodity hardware. We develop programs that engage users mentally and collect user's physiological data. Our setup consists of a desktop application that presents the user with tasks of varying difficulty that we call Cognitive Load Test, another desktop application that runs in parallel and through a side-task aims to gauge the user's cognitive engagement - Secondary Task, a wearable sensor that through a mobile phone app collects data about the user's physiological signals, and pre- and post-test questionnaires that evaluate the user's overall cognitive capacity - N-back test and Hexaco personality test. In the thesis we provide detailed explanations of the testing environment, especially the parts that were originally developed for the thesis – the secondary task desktop application, the web-based pre/post questionnaires and tests, and the database orchestration – and report on the experiences with real-world experiment we conducted with 27 volunteers. We first review related prior work and then describe different components, how we designed them and implemented them. We explain which data we collected and why we think it is useful to collect that data.

Chapter 2

Related Work

Our work subscribes to the definition of *cognitive load theory* by Paas et al. which defines cognitive load as a multidimensional construct that represents the load imposed on a learner's cognitive system by performed task. [20] There are three different types of cognitive load, namely *intrinsic*, *extraneous (ineffective)* and *germane (effective)* load. Because different types of load are additive - they stack - and we cannot affect intrinsic load, we must try to minimize extraneous load that results from poor instructions. Our framework uses tasks that are described later, to try to spike germane load and then measure the load as a whole. Paas defines *mental effort* as an aspect of cognitive load that refers to the cognitive capacity that is allocated in order to meet demands imposed by the task. As such, it represents cognitive load as a whole and has been traditionally been measured with subjective self-evaluation such as NASA-TLX [15]. Another aspect of cognitive load that can be measured is *Performance*, which is defined as achieved performance on a task based on the number of correct test items, number of errors, and time spent on task [7].

One of the goals of the framework is to see if we can measure mental effort with physiological data collected while a task is performed. In order to design a good framework, we first need to make a short overview of other studies that dealt with measuring cognitive load. Jaeggi et al. used functional mag-

netic resonance imaging to investigate the difference in activation of selected cortical areas while users performed challenging dual task [17]. One task was a variant of an N-back test that our framework also employs and is described in detail later on. The second task added stimuli to which users had to react. They measured the accuracy of the N-back test and reaction time needed to respond to the stimuli. fMRI showed that high performers activated fewer regions of their brains in contrast with low performers. Data also showed that with more difficult N-back task reaction time also increased.

Fritz et al. collected EDA (electrodermal activity) data, EEG data and eye tracker data on programmers working on different tasks and tried to classify those tasks as either easy or hard [11]. While eye tracking sensor performed the best, EDA showed similar performance to EEG while being much less invasive and much cheaper. Note that they still did not use commodity devices for EDA sensing.

Gjoreski et al. used a wristband for stress detection [12]. The wristband hosts a multitude of different sensors such as HR (heart rate), BVP (blood volume pulse), GSR (galvanic skin response), ST (skin temperature), RR interval (time between successive R points in QRS complex) and accelerometer. This was a non-intrusive method for detecting stress that could also be useful to measure cognitive load.

Our approach builds upon Psycho-Physiological Measures for Assessing Cognitive Load [14]. The authors developed an application that presented users with six different types of tasks in order to stimulate them and impose cognitive load. They monitored the subjects with an eye tracking device, HR band, EEG headset and armband which collected GSR and ECG data. They relied on measuring performance between easier and harder task and on self-evaluation after each task.

Prior studies relied on intrusive and expensive techniques which are not scalable to billions of users. Our work focuses to find out if cheap and accessible hardware, such as mobile phones, fitness wristbands, smart watches and other wearable technology, could replace those expensive techniques,

how reliable it is, and what obstacles lie on the path to achieve unobtrusive cognitive load inference.

Chapter 3

Cognitive Load Experimentation Framework

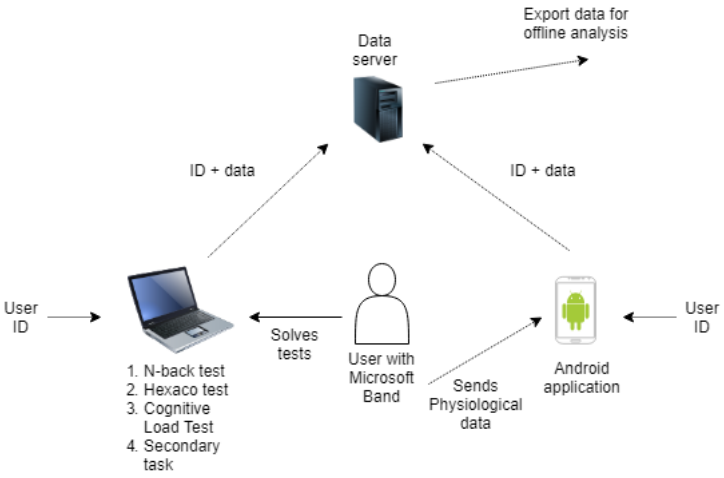

Figure 3.1: Parts of the framework and how they are connected.

This chapter will describe the experimentation framework, its components and the data that was collected. It also describes reasoning behind the protocol and why we collected certain types of data.

3.1 Framework Components

3.1.1 N-back Test

Because people have different cognitive load capacity [17] we wanted to measure each subject's load capacity. N-back test gives us relative cognitive load capacity of the subject which can help us better understand Cognitive Load Test results in light of multiple users.

The idea of a N-back test is to output a stimulus that the user must remember N steps back and with each step the remembered stimuli need to be updated. How well the user performs is measured by how many times in the given time the user successfully detects encountering the same stimulus as N steps back. There are many different versions of N-back tests. Our version is different from the one used in [17] and is explained in detail in chapter about implementation.

Because we would like to discern between levels of cognitive load, we tested subjects for cognitive capacity with N-back test. With this data we would likely be able to tell if user was under or over stressed during the cognitive load test and thus understand physiological data better.

N-back test lasts 2 minutes each (2-back and 3-back) and we measure how many answers are correct, how many answers are wrong and how many answers are missed. Higher scores meaning larger cognitive capacity.

3.1.2 Hexaco Personality Test

Hexaco personality test gives us the measures of the 6 major personality dimensions, namely Honesty-Humility, Emotionality, Extraversion, Agreeableness (versus Anger), Conscientiousness and Openness to Experience [5].

We do not have any firm hypotheses. However, collecting these data enables us to reveal potential links between the personality traits and the cognitive load test results. The subjective ratings of test difficulties might be particularly sensitive to these traits.

3.1.3 Cognitive Load Test

Cognitive load test was adapted from [14], where the authors found positive results that cognitive load can indeed be inferred from physiological data. Their findings also note that different difficulty ratings of the tasks (easy, medium, hard) indeed result in different task load index and completion times. Test consist of six different elementary cognitive tasks that measure three contextual factors (speed of closure, flexibility of closure and perceptual speed) [10]. Six different tasks are:

- Hidden Pattern test where the user is presented with a figure and must detect the pattern of the presented figure in other figures that contain similar patterns where one of which masks the presented pattern.
- Gestalt Completion test where the user is shown an incomplete drawing and must identify what is in the drawing.
- Finding A's where the user must find words that contain the letter 'a'.
- Number Comparison test where the user must detect long numbers that are the same.
- Pursuit test where the user must follow overlapping irregularly curved lines with his eyes and connect numbers on the left side to the letters on the right side.
- Scattered X's test where the user must find letter 'x' on the screen among other scattered letters.

After each task there is a NASA-TLX questionnaire [19] where the user reports subjective feelings about the task. The test is in the form of a computer application which presents tasks to the users and logs the users' actions. The log file can then be analysed and from there correctness and speed can be determined. The log file also denotes in which stage the test currently is so secondary task can read the file and trigger when needed. We localized the tests used to induce cognitive load and added an additional parallel

task that we call secondary task. We also added an additional screen to the application where the secondary task is shown and explained to the user.

3.1.4 Secondary Task

In parallel with cognitive load test we added a secondary task. The secondary task is a task that is administered in parallel to the primary task, with the goal of "filling up" vacant cognitive capacities of the user. The performance on the secondary task is expected to reflect the amount of these vacant resources, thus indirectly determining the cognitive load devoted to the primary task. Pass et al. [20] in his overview of the studies on cognitive load notes that secondary task technique has been rarely used in research. Most likely because of the danger that it impacts the primary task. We found the premise worth exploring so we decided to include this technique in our framework.

Secondary task measured response times to a visual stimulant of the test subject during the cognitive load test. A black square appeared randomly in one corner of the screen. At first opacity was set low so that the square was see through but then with time the square became more and more solid and clearly visible. The subject had to detect the square as soon as possible and click on it. This parallel task gives us information about how hard the cognitive load tasks actually were so we do not rely only on self-assessment of the test subjects. Longer response times should mean harder, more demanding cognitive load task. A similar secondary task was already used by DeLeeuw et al. [9]. The authors found that the load that affects the task's performance the most is extraneous load. This is another thing that should be taken into consideration.

3.1.5 Android Application for Physiological Data Collection

An android application was developed by the Jožef Stefan Institute. The application connects to a fitness wristband (Microsoft Band 2) and records physiological data on a mobile phone. Data is then timestamped and uploaded to a server along with user ID and can be then analysed off-line.

Physiological data that is collected is RR intervals (time between heart beats, also known as heart rate variability), Galvanic Skin Response, Skin Temperature, barometer data, accelerometer data and data from the UV sensor.

3.1.6 Demographics Questionnaire

Demographic data collected consisted of subject's age, gender, handedness and level of education. We believe it is likely that these variables could have an impact on performance and it would be interesting to see how they correlate with other data.

Chapter 4

Design and Implementation of Framework Components

In this chapter we describe the components that we developed in more detail. We also explain design considerations and implementation details. Design of the components impacted the collected data directly that is why in some cases we developed more than one version and parameterized the software in order to test different approaches and configurations.

N-back test and Hexaco personality test are implemented as a web page. Secondary task is implemented as a desktop application based on Qt framework.

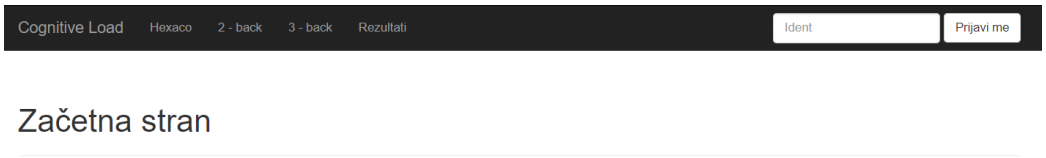

Figure 4.1: The starting page of the web application from where a user can input ID and choose between tests.

4.1 N-back Test

4.1.1 Description

N-back test consist of a 3 by 3 square board. Periodically random squares are coloured black. The subject's task is to remember positions of the coloured squares N iterations backwards and then detect when position of the current coloured square is the same as it was N iterations ago. We will call this a "win condition". For example, in 2-back task, if we have a coloured square at position 1, then at position 4 and then again at position 1 again this means that coloured square is at the same position (square 1) as was two iterations ago. Detecting this would increase the subject's *correct* counter. If not, *correct* counter would not be incremented thus reducing ratio of *correct* versus *all possible*. If subject would detect that the coloured square is at the same position incorrectly, *incorrect* counter would be incremented. The subject is also informed if his/her guess was correct or incorrect via text flash alongside the check button when pressed.

4.1.2 Implementation

The N-back test was implemented as a Web application in JavaScript. Application draws a 3 by 3 square board on the screen and colours squares black at random. The user interacts with the application via buttons. Application was architectured as a game with a main loop that managed timeouts for callbacks which coloured in the squares and erased the board at correct intervals. The most challenging part of the implementation was managing application state because of the asynchronous nature of the JavaScript execution. A lot of time went into testing that button presses did not corrupt the "game" flow, that double presses did not increment *correct* and *incorrect* counters more than once and similar problems.

Time during which a square is coloured in and time when no square is coloured are called stimulus time and interstimulus interval, respectively. We

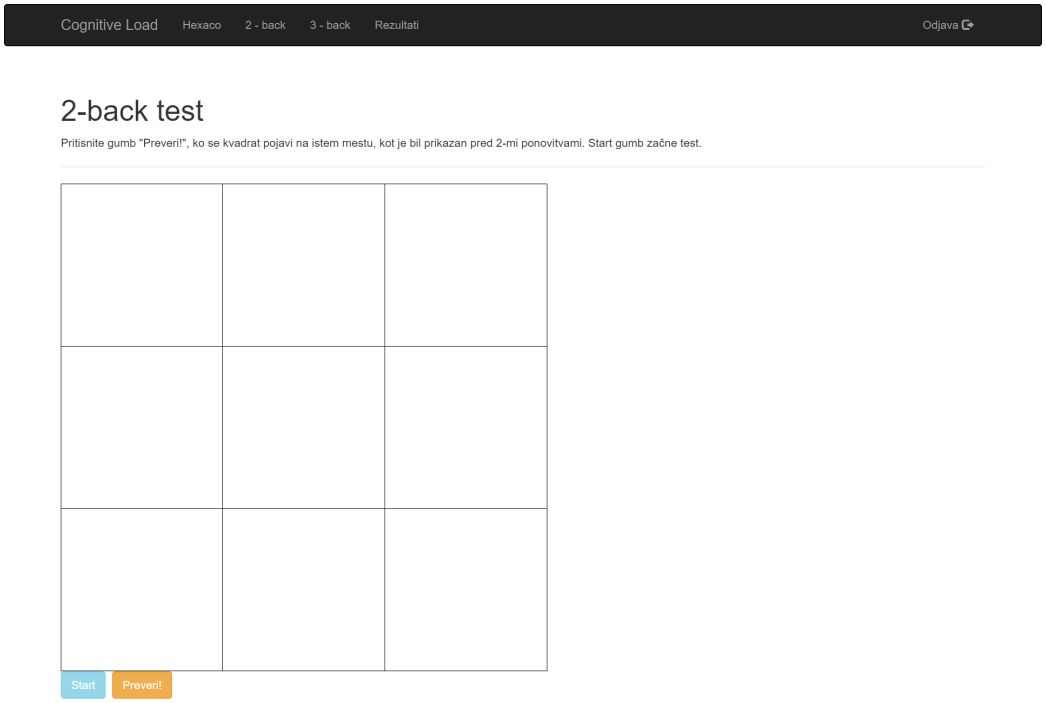

Figure 4.2: Board for N-back test with buttons to start the test and to check if our detection of winning condition is correct or not.

set them to 500ms for stimulus time and to 2500ms for interstimulus interval as recommended by [17].

One of the most important parts of the application is the random position generation. At first the algorithm we used generated one of the 9 positions at random with equal probability. "Playtesting" (going through the test many times) showed us that when the algorithm produced the same positions sequentially it was prohibitively difficult to follow the flow of the game. With regard to this observation, we excluded the possibility of generating the same position one after another. Playtesting also showed us that the "win condition" occurred with a far too low probability. That is why we first check for the win condition with 25% probability. If the win condition occurs, we return the same position as was N-back intervals ago. If win condition does not occur, we return one of the 8 possible positions at random. This makes

that the win condition occurs with a probability of 0.25 + 1/8.

Because at the start of the game the winning position is at position 0 (not enough squares coloured yet to be possible for N-back win condition to occur) we included this check for completion so that we do not return position 0 and with that disrupt the game flow.

```
1  function gen_number(last) {
2      var matching_number = window.number_history[window.
           number_history.length - num_back];
3      if (matching_number != 0) {
4          var repeat = Math.floor(Math.random() * 100) + 1;
5          if (repeat < chance_to_succeed) {
6              return matching_number;
7          }
8      }
9      var n = Math.floor(Math.random() * 9) + 1;
10     while (n == last) {
11         n = Math.floor(Math.random() * 9) + 1;
12     }
13     return n;
14 }
```

Listing 4.1: Code showing random position generation

4.1.3 Test Results

At the end, the test provides us with three different measurements of subject's performance. Number of *correct* detections, number of *incorrect* detections and the number of *all possible* detections for each version of the test (2-back and 3-back). Results proxy users' overall cognitive capacities.

4.2 Hexaco Test

4.2.1 Description

Hexaco questionnaire was implemented as a web application. User starts the test and is presented with a statement and then has to choose one of the five possible options depending on how much he or she agrees or disagrees with the given statement. User chooses one of the options by clicking the appropriate button as shown in Figure 4.4. At the end of the test, after the user has responded to all 60 statements, results are shown and can be printed out.

4.2.2 Implementation

Programming language Python was used for this task as it offers powerful and expressive features while maintaining simplicity of use. Flask micro-framework provided abstractions such as routing the URLs, parsing HTTP protocol and converting it to the Request object and gluing together libraries that provided templating engine (Jinja), form data validation (WTForms) and representation of the database as models (SQLAlchemy). This approach is known as MVC (Model View Controller) pattern. It separates the different concerns, making development more manageable.

Model part exposes the database via ORM (Object-Relational Mapper). Instead of dealing with table data and writing SQL directly, ORM provides a Python object that is a representation of a table data. It also offers support for querying and modifying the data, eliminating the need to write repetitive SQL statements.

```
1 class User(db.Model):
2     id = db.Column(db.Integer, primary_key=True)
3     ident = db.Column(db.String(5), index=True, unique=True)
4     hexaco = db.Column(db.String(119), nullable=True)
5     nback_2 = db.Column(db.String(120), nullable=True)
6     n2_time = db.Column(db.Integer, nullable=True)
```

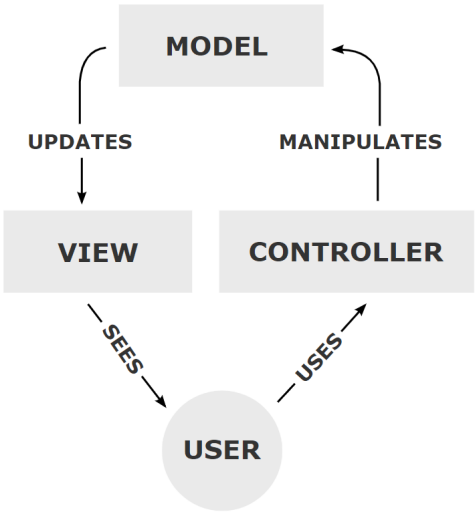

Figure 4.3: MVC information flow diagram

```
7     n3_time = db.Column(db.Integer, nullable=True)
8     nback_3 = db.Column(db.String(120), nullable=True)
```

Listing 4.2: Definition of the User model

```
1 user.hexaco = '|'.join(str(session['answers'][str(i)]) for i
    in range(1, len(QUESTIONS) + 1))
2 db.session.add(g.user)
3 db.session.commit()
```

Listing 4.3: Recording Hexaco answers for a user represented as a string separated by pipe character

View part of the MVC are HTML templates. The following listings show a Python form that gets rendered in a Jinja template along with a question. Templates enable us to separate representation from business logic. It also enables us to generate dynamic content in a simple way that is easy to maintain.

```
1 class HexacoQuestionForm(FlaskForm):
2     one = SubmitField(label='se močno ne strinjam')
3     two = SubmitField(label='se ne strinjam')
4     three = SubmitField(label='sem nevtralen')
```

```
5       four = SubmitField(label='se strinjam')
6       five = SubmitField(label='se močno strinjam')
```

Listing 4.4: Form that represents user choices under each question

```
1 <div class="col-md-12">
2     <p>{{ question_number }}/60</p>
3     <h3>{{ question }}</h3>
4     <div class="input-group-btn" role="group">
5         {{ wtf.quick_form(form) }}
6     </div>
7 </div>
```

Listing 4.5: Part of the hexaco_questions.html

Templates then get rendered in the browser as depicted in Figure 4.4.

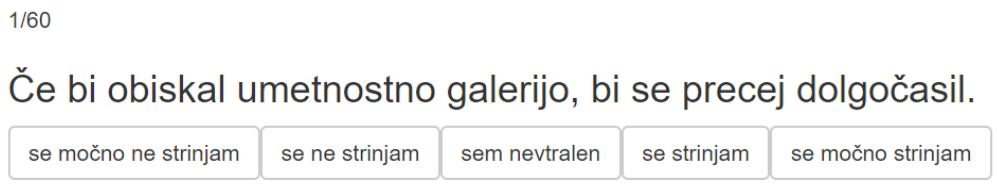

Figure 4.4: Question with available choices

The last part of MVC pattern is the Controller. In Flask terminology this is named View, not to be confused with View in the MVC. View is a function that gets called based on the URL. In that function, a Request object is available and parameters from URL route are extracted. In the following listing we can see a view that gets called on URL http://example.com/hexaco/16 for example with allowed methods GET and POST (line 1). It gets question number as a parameter (line 3). If question number is larger than the number of questions (line 4), we save the results that were stored in the session and redirect (line 8) the user to the results page. If not, we create a form and validate it on submit (line 10). This means that if method is POST, it automatically takes data from Request object, fills in the form and validates it. If form is valid, we store the answers in the session (line 13 and

14) and redirect the user to the next question (line 16). If form is not valid, this means we received a GET request and we render the template for an appropriate question (line 19).

```
1  @app.route('/hexaco/<int:question_number>', methods=['GET', '
       POST'])
2  @login_required
3  def hexaco_questions(question_number):
4      if question_number > len(QUESTIONS):
5          g.user.hexaco = '|'.join(str(session['answers'][str(i
               )]) for i in range(1, len(QUESTIONS) + 1))
6          db.session.add(g.user)
7          db.session.commit()
8          return redirect(url_for('results'))
9      form = HexacoQuestionForm()
10     if form.validate_on_submit():
11         for i, button in enumerate([form.one, form.two, form.
               three, form.four, form.five], start=1):
12             if button.data:
13                 session['answers'][str(question_number)] = i
14                 session.modified = True
15                 break
16         return redirect(url_for('hexaco_questions',
               question_number=question_number + 1))
17
18     question = QUESTIONS[question_number]
19     return render_template('hexaco_questions.html', question=
           question, form=form, question_number=question_number)
```

Listing 4.6: View that renders the questions and handles POSTed data

Apart from structuring the application in MVC pattern, one must also take care when handling POST request. GET request do not change the state of the application and are therefore safe by default. POST request, however, changes the application state and must be handled properly. Because web server is multi-threaded it can handle multiple requests at once. This is a problem in a situation where the user clicks buttons in rapid succession before the first request has had a chance to return. If we do not handle those

requests carefully, questions could be skipped and answers could be recorded incorrectly. That is why when saving answers special care is taken that this does not happen. Also, after every POST request there is a redirect that prevents resending of the data on page refresh.

4.2.3 Test Results

For each statement the users' response is saved as an integer from 1 to 5. "I strongly disagree" option counts for 1 and "I strongly agree" counts for 5. Then for each of the personality dimensions honesty-humility, emotionality, extraversion, agreeableness (versus anger), conscientiousness and openness to experience a score is calculated based on a scoring key that specifies which statements affect which dimension. Final score is a mean across all statements that affect a dimension.

4.3 Secondary Task

4.3.1 Description

Secondary task is implemented as a desktop application. It reads a file in which the Cognitive Load Test application writes its state in and triggers challenges when CLT application reaches predefined states. The challenge is a square that appears randomly in one of the four corners. At first the square is opaque, as seen in Figure 4.5, and then gradually becomes solid black, shown in Figure 4.6. The user has to react and click on the square as fast as possible.

4.3.2 Design and Ideas

Secondary task demanded along with implementation also a design consideration. Multiple ideas were suggested and implemented to see and test out how they work together with the primary test. The secondary task was also designed to fit in with the real-world study that is explained in Section 6.2

(Future Work). Three ideas were considered, two implemented and only one was verified in our iterative design process.

The idea behind the secondary task is that it emits a stimulus and user needs to react to that stimulus. At first we thought of using audio signals but this idea was dropped because of the real-world study where we cannot control the environment ergo making it impossible to take the test without speakers or headphones, and making it especially challenging in a noisy environment. The second idea was that the secondary task randomly displayed numbers from 1 to 9. Whenever three consecutive numbers have the sum of 15 or more, the user should click a button. This idea was implemented and tested but determined that it was too difficult in parallel with the primary task. It could be that this secondary task would fill up the cognitive work load capacity and with that we would not see any difference in measurements between easy, medium and hard primary tasks. Other similar ideas with calculating numbers or keeping things in working memory were dropped because of the same reasoning. This is why we went with the opaque squares and measured the reaction time instead of loading additional work on the user.

4.3.3 Implementation

The secondary task is implemented in Python as a Windows application with PyQt library. Qt framework enables us to display opaque windows without borders, helps with handling events such as mouse presses and displaying a system tray icon.

The final version of the secondary task was a multi-threaded application. One thread tailed the logging output from the primary task app and based on that it was switching secondary task on and off. Another thread set the visibility of the colour rectangle at a random interval and changed opacity. The main thread handled user input, namely user clicking the square.

But before implementing the final version we needed a way of testing different approaches and parameters. In the development version of the appli-

cation the configuration file was read and from this config it was determined which task to run and with what parameters. For example, with the colour rectangle you could change the minimum time between tasks (low range) and the maximum time between tasks (high range). The task was then randomly triggered on an interval between low and high range. Additionally, the developer could specify how fast opacity changed once the square was displayed with opacity step (how much opacity changed in one step, where 0 is totally opaque and 1 is solid) and opacity time (time between opacity change steps in seconds).

```
1  [Main]
2  task = color-rectangle
3  low_range = 1
4  high_range = 10
5
6  [ColorRectangle]
7  width = 300
8  height = 300
9  opacity_step = 0.01
10 opacity_time = 0.3
```

Listing 4.7: Example of the configuration file

This enabled us to test different settings easily without changing the source code and rebuilding and re-installing the application each time.

4.3.4 Test Result

Each activation of the secondary task is logged into a text file so we know the time when the square appeared on the screen and when the user clicked on it. From these logs we can calculate response times and link activation to the primary task that was active during that time.

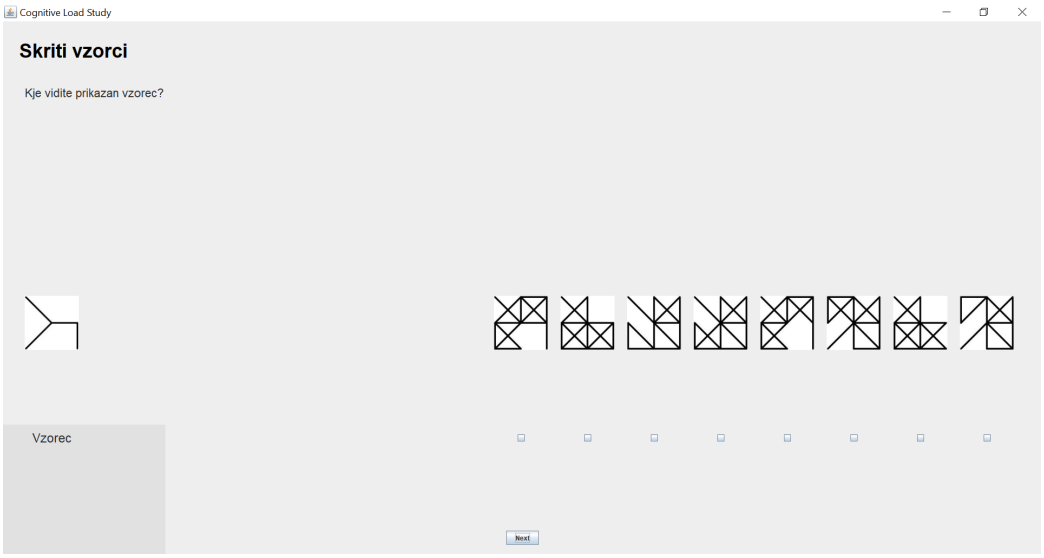

Figure 4.5: One of the tests from primary task application with secondary test square (lower left corner) shortly after appearing and thus very opaque.

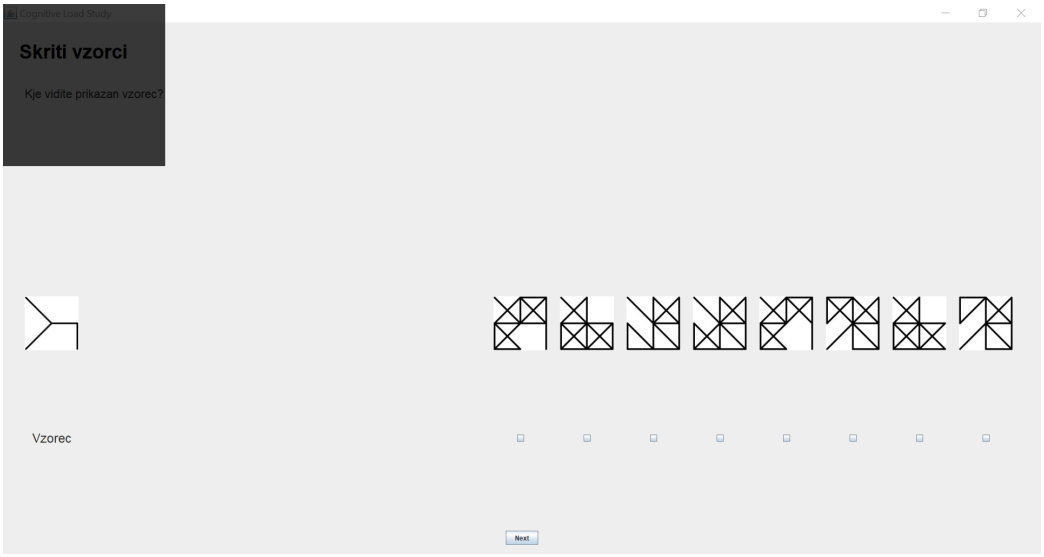

Figure 4.6: Secondary test square (upper left corner) almost completely solid.

Chapter 5

User Study

Our framework was used by two studies [13] [18]. Specifically, [13] was interested if cognitive load estimation is possible with cheap wearable sensing devices, so the framework was built with this study in mind and we conducted user testing for it. In this chapter we will describe this study and how the framework was used.

5.1 Background and Overview

For the user study we recruited 27 volunteer test subjects. Recruiting was done via faculty's mailing list, social networks and word of mouth. Volunteers were provided with a quiet and interruption free environment. They were sat before the computer and fitted with Microsoft Band 2 fitness wristband. Wristband was connected to an Android phone running the application mentioned in chapter 3. There was one supervisor present that clarified any questions test subjects might have and ensured that each test followed the specified protocol.

We collected demographic data before the test with half of the subjects and after the test with the other half. This is because we were interested if collecting the demographic data before the test somehow affects performance. Test subjects might think that a variable like age is important and that it

would affect performance via self-doubt if the subjects think they are too old for example.

Then users were presented with N-back instructions that were printed on paper. When everything was clear, the users solved N-back tests on the computer with a 3-minute pause in between in order for physiological signals to return to normal. After the N-back tests the users solved the personality test which required minimal mental effort so the users were prepared for the next stage.

Now followed the main part which was also the most taxing on the users. Cognitive Load Test was started and the secondary task along with it. Cognitive Load Test consisted of 6 cycles. Each cycle challenged users with a different task described in Chapter 3. In each cycle there were three such tasks of increasing difficulty. After each task there was a self-evaluating questionnaire and a rest period. Instructions on how each task type is solved were included in the application itself before each cycle along with one solved example.

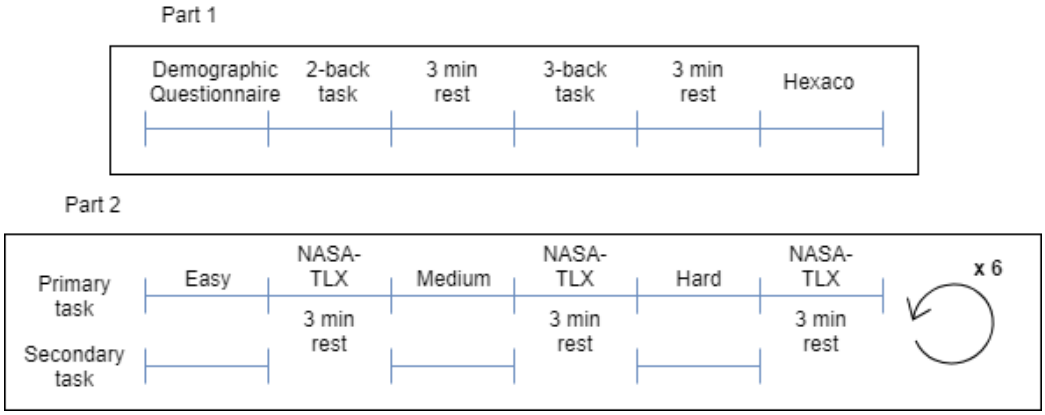

Figure 5.1: Steps that the subject goes through while being tested.

5.2 Test Protocol

Supervisors of the user testing were using protocol in order to ensure consistent execution. It was also very important to remember to change ID for each user in the applications so the data could be linked together. The protocol was as follows:

1: Pick one of the pre-generated IDs for the test subject.

2: Based on the ID we collect the demographic data or we skip to the next step.

3: Test subject is equipped with the Microsoft Band 2. Mobile application for collecting data from the wristband is started and the ID is put in the application.

4: Log in the web application with the ID.

5: Test subject is provided with instructions for the N-back test. Clarifications are made by supervisor if needed.

6: Subject solves the 2-back test, followed by 3 minutes of rest.

7: Subject solves the 3-back test, followed by 3 minutes of rest.

8: Subject solves the Hexaco personality test.

9: Cognitive load test and secondary task are started with the ID.

10: Subject gets familiar with the secondary task.

11: Subject solves the cognitive load test.

12: We collect demographic data if we did not do that already.

13: Test subject gets print outs of the web application test results.

5.3 Experiences

Testing went well for the most part, however, there were some bugs and annoyances discovered along the way. We describe some of these in this section.

Mobile application that collected Microsoft Band 2 data needed to be synced occasionally with the server during the test. If the test took too long, data points could get lost as mobile phone could not store them all. Syncing

should be done between users reliably if not, data points could be assigned to a different user. This got corrected by assigning data points to a user based on time stamp. Better GUI with syncing status and progress would help here.

The task where users needed to find 5 words that contain multiple 'a' characters in them was found to contain more than 5 such words. Consequentially, users completed this one faster or found more than 5 words. This was checked for correctness by a human but if it had been tested programmatically by writing a software unit test, this could be avoided.

Some tasks could have clearer instructions or should emphasize rules better. For example, a task where users tracked convoluted lines from one point to another with only their eyes should express more strictly that tracking with the help of a finger is not allowed. This was solved by having a supervisor conducting the test, ensuring tests were completed fairly and consistently.

An application log parsing for results should be designed and tested before the tests. This led to some more work at the end of the testing because logs were badly designed for parsing and data extracting was harder to do.

Regarding user experience, some were annoyed because they had to rest between tasks for physiological data to return back to normal after working on the task and they were eager to solve the next task. They were also annoyed by having to self-evaluate after each task. However, the whole testing experience was positive for them as tasks were interesting and challenging to do.

We recruited and performed tests with 27 volunteers but because of some technical issues with syncing physiological data from the wristband to the server only 21 users provided valid data. The table below shows a summary of demographic information.

Total number of subjects: **27**
Age:

20-30	31 - 50	51+
19	7	1

Gender:

Male	Female
22	5

Education:

Highschool	Bachelor	MsC	PhD
7	7	9	4

Three different types of physiological signals were collected (R-R intervals, Galvanic Skin Response and Skin Temperature) from which a multitude of statistical features can be extracted. For example, GSR signal can be split into fast acting component and slow acting component. Further, we can calculate mean, standard deviation, quartile deviation, 1st and 3rd quartile and other statistical features.

Chapter 6

Conclusion and Future Work

We developed a framework for collecting data and measuring cognitive load. The framework consists of multiple applications that were made to work together via user IDs and timestamps: Cognitive Load Test, Secondary task, N-back and Hexaco tests and an Android application for physiological data collection. Secondary task, N-back test and Hexaco test were designed and developed specifically for this framework. The code can be found on GitHub website here [4] and here [3].

There have been two studies performed with the help of our framework. For the one study that we described we also specified the testing protocol, recruited users and administered the tests.

Developing the framework was an iterative process of researching what other researchers used before, what worked and what did not. Then ideas were implemented and tested. This cycle was repeated until we were pleased with the results. With each iteration design flaws were corrected and software bugs fixed. For the final test of the framework we administered the experiments with real users which gave us valuable feedback that the framework is indeed useful.

6.1 Lessons Learned

Our testing environment had a lot of different software and hardware pieces. A good strategy was needed to connect different pieces and link the data together. ID and time-based identification proved to be adequate in ensuring that data was assigned correctly to each tested user.

Easy to use and well tested tools are important in order for testing to be successful and without problems. When problems arose, it was because the tools were not easy to use or they were not properly tested. Much of the testing should be automated or should follow clear and well-defined procedure in order to minimize chances of human error and therefore ensure that tests are done in a controlled way in order to provide more reliable data.

The ability to quickly iterate on the design and change parameters of the tools is important in order to quickly test out different ideas and verify them if they work or not. With testing, the developer gets valuable feedback and insight and can then improve on the design or see when the design is inappropriate and go to the next idea.

6.2 Future Work

The collected data needs to be statistically analysed in order to test the hypotheses for which this test environment was designed and implemented. Data outliers, if there are any, will probably show us where errors in our testing environment were hidden, but hopefully there is enough good data to reliably test the hypotheses.

After data analysis, the environment has also the option to be extended for use in the real-world tests. With little adaptation, people at home or work could be equipped with wristbands that could collect data while they solve their specific real-world tasks. A secondary task would be installed on their computer and would be remotely triggered at set intervals. The phone application could be extended to also collect data from the users' environment

with different available sensors such as noise level, brightness, etc.

6.3 Acknowledgements

We would like to thank Timotej Strnad who provided us with Slovene translation of the Hexaco test. Translation is not a simple matter of translating words but has to be first methodically verified and then approved by authors.

Bibliography

[1] How many times are people interrupted by push notifications? https://askwonder.com/q/how-many-times-are-people-interrupted-by-push-notifications-58efcbf59682ca280093ebd9, 2015. (Acquired 28.09.2018).

[2] Number of mobile phone users worldwide from 2015 to 2020. https://www.statista.com/statistics/274774/forecast-of-mobile-phone-users-worldwide/, 2018. (Acquired 28.09.2018).

[3] Github - Code repository for N-back and Hexaco tests. https://github.com/mfrlin/cognitive_load_web, 2019. (Acquired 6.2.2019).

[4] Github - Code repository for Secondary task. https://github.com/vpejovic/cognitiveload, 2019. (Acquired 6.2.2019).

[5] THE HEXACO PERSONALITY INVENTORY - REVISED. http://hexaco.org/, 2019. (Acquired 21.01.2019).

[6] Christoph Anderson, Isabel Fernanda Hübener, Ann-Kathrin Seipp, Sandra Ohly, Klaus David, and Veljko Pejovic. A survey of attention management systems in ubiquitous computing environments. *CoRR*, abs/1806.06771, 2018.

[7] Brian P Bailey and Joseph A Konstan. On the need for attention-aware systems: Measuring effects of interruption on task performance, error rate, and affective state. *Computers in human behavior*, 22(4):685–708, 2006.

[8] Jelmer P Borst, Niels A Taatgen, and Hedderik van Rijn. What makes interruptions disruptive?: A process-model account of the effects of the problem state bottleneck on task interruption and resumption. In *Proceedings of the 33rd annual ACM conference on human factors in computing systems*, pages 2971–2980. ACM, 2015.

[9] Krista E DeLeeuw and Richard E Mayer. A comparison of three measures of cognitive load: Evidence for separable measures of intrinsic, extraneous, and germane load. *Journal of educational psychology*, 100(1):223, 2008.

[10] Ruth B Ekstrom, John W French, and Harry H Harman. Cognitive factors: Their identification and replication. *Multivariate Behavioral Research Monographs*, 1979.

[11] Thomas Fritz, Andrew Begel, Sebastian C Müller, Serap Yigit-Elliott, and Manuela Züger. Using psycho-physiological measures to assess task difficulty in software development. In *Proceedings of the 36th International Conference on Software Engineering*, pages 402–413. ACM, 2014.

[12] Martin Gjoreski, Hristijan Gjoreski, Mitja Luštrek, and Matjaž Gams. Continuous stress detection using a wrist device: in laboratory and real life. In *Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing: Adjunct*, pages 1185–1193. ACM, 2016.

[13] Martin Gjoreski, Mitja Luštrek, and Veljko Pejović. My watch says i'm busy: Inferring cognitive load with low-cost wearables. In *Proceedings of the 2018 ACM International Joint Conference and 2018 International Symposium on Pervasive and Ubiquitous Computing and Wearable Computers*, UbiComp '18, pages 1234–1240, New York, NY, USA, 2018. ACM.

[14] Eija Haapalainen, SeungJun Kim, Jodi F Forlizzi, and Anind K Dey. Psycho-physiological measures for assessing cognitive load. In *Proceed-*

ings of the 12th ACM international conference on Ubiquitous computing, pages 301–310. ACM, 2010.

[15] Sandra G Hart and Lowell E Staveland. Development of nasa-tlx (task load index): Results of empirical and theoretical research. In *Advances in psychology*, volume 52, pages 139–183. Elsevier, 1988.

[16] Eric Horvitz, Johnson Apacible, and Muru Subramani. Balancing awareness and interruption: Investigation of notification deferral policies. In *International Conference on User Modeling*, pages 433–437. Springer, 2005.

[17] Susanne M Jaeggi, Martin Buschkuehl, Alex Etienne, Christoph Ozdoba, Walter J Perrig, and Arto C Nirkko. On how high performers keep cool brains in situations of cognitive overload. *Cognitive, Affective, & Behavioral Neuroscience*, 7(2):75–89, 2007.

[18] Tilen Matkovič and Veljko Pejović. Wi-mind: Wireless mental effort inference. In *Proceedings of the 2018 ACM International Joint Conference and 2018 International Symposium on Pervasive and Ubiquitous Computing and Wearable Computers*, UbiComp '18, pages 1241–1249, New York, NY, USA, 2018. ACM.

[19] Abhinav Mehrotra, Mirco Musolesi, Robert Hendley, and Veljko Pejovic. Designing content-driven intelligent notification mechanisms for mobile applications. In *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, pages 813–824. ACM, 2015.

[20] Fred Paas, Juhani E Tuovinen, Huib Tabbers, and Pascal WM Van Gerven. Cognitive load measurement as a means to advance cognitive load theory. *Educational psychologist*, 38(1):63–71, 2003.